案例 20-多边形 Sutherland-Hodgman 裁剪算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

说明：本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

一、知识点

Sutherland-Hodgman 裁剪算法原理：Sutherland-Hodgman 裁剪算法是用裁剪窗口的四条边依次对多边形进行裁剪，裁剪算法的输出结果为裁剪后多边形顶点序列。（Sutherland-Hodgman 裁剪算法裁剪凹变形时存在错误，Weiler-Atherton 算法可以解决这个问题，Weiler-Atherton 算法思路见第六点知识点补充。）

二、案例描述

本案例使用多边形 Sutherland-Hodgman 裁剪算法，对有七个顶点的多边形进行裁剪。

三、实现步骤

1. 在 CTestView 中添加绘制裁剪窗口函数、窗口边界赋值函数、裁剪多边形函数、计算交点函数、坐标系转换函数。

2. 在 CTestView 中添加裁剪相关功能键的消息响应函数，在 OnDraw 中调用 DoubleBuffer 函数定义客户区坐标。

四、主要算法

CTestView 类

```
public:
    void ReadPoint();//读入点表函数
    void DoubleBuffer(CDC* pDC);//双缓冲函数
    void DrawWindowRect(CDC* pDC);//绘制裁剪窗口函数
    void DrawObject(CDC* pDC, BOOL bClip);//绘制多边形函数
```

```
    void ClipBoundary(CP2i Rect0, CP2i Rect1);//窗口边界赋值函数
    void ClipPolygon(CP2d *out, int Length, UINT Boundary);//裁剪多边形函数
    BOOL Inside(CP2d, UINT);//在窗口边界处判断函数
    CP2d Intersect(CP2d p0, CP2d p1, UINT Boundary);//计算交点函数
    CP2i Convert(CPoint point);//坐标系变换函数
protected:
    int nClientWidth, nClientHeight;//屏幕客户区宽度和高度
    int nHWidth, nHHeight;//屏幕客户区的半宽和半高
    CLine *line;//直线的指针
    CP2d *In, *Out;//裁剪后的顶点数组中的顶点个数
    CP2i Rect[2];//窗口顶点数组
    int OutCount;//裁剪后的顶点数组中的顶点个数
    int RtCount;//窗口顶点个数
    int Wxl, Wxr, Wyb, Wyt;//裁剪窗口边界
    BOOL bDrawRect;//是否允许绘制窗口
    BOOL bClip;//是否允许裁剪
void CTestView::DoubleBuffer(CDC* pDC)
{
    CRect rect;//定义客户区
    GetClientRect(&rect);//获得客户区的大小
    nClientWidth = rect.Width();//屏幕客户区宽度
    nClientHeight = rect.Height();//屏幕客户区高度
    nHWidth = nClientWidth / 2;//屏幕客户区半宽
    nHHeight = nClientHeight / 2;//屏幕客户区半高
    CDC memDC;
    memDC.CreateCompatibleDC(pDC);
    CBitmap NewBitmap, *pOldBitmap;
    NewBitmap.CreateCompatibleBitmap(pDC, nClientWidth, nClientHeight);
    pOldBitmap = memDC.SelectObject(&NewBitmap);
    memDC.FillSolidRect(&rect, pDC->GetBkColor());
    if (RtCount && !bClip)
        DrawWindowRect(&memDC);//绘制窗口
    DrawObject(&memDC, bClip);////绘制多边形
    pDC->BitBlt(0, 0, nClientWidth, nClientHeight, &memDC, 0, 0,
SRCCOPY);//将内存位图拷贝到屏幕
    memDC.SelectObject(pOldBitmap);//恢复位图
    NewBitmap.DeleteObject();//删除位图
}
void CTestView::ReadPoint()
{
    In = new CP2d[InMax];//输入顶点表
    In[0].x = 50;  In[0].y = 100;
    In[1].x = -150; In[1].y = 300;
    In[2].x = -250; In[2].y = 50;
```

```
    In[3].x = -150; In[3].y = -250;
    In[4].x = 0;   In[4].y = -50;
    In[5].x = 100; In[5].y = -250;
    In[6].x = 300; In[6].y = 150;
    Out = new CP2d[OutMax];//输出顶点表
    for (int i = 0; i < OutMax; i++)
        Out[i] = CP2d(0, 0);
}
void CTestView::DrawWindowRect(CDC* pDC)//绘制裁剪窗口函数
{
    CRGB LineClr = CRGB(0, 0.5, 0);
    line->MoveTo(pDC, nHWidth + Rect[0].x, nHHeight - Rect[0].y, LineClr);
    line->LineTo(pDC, nHWidth + Rect[1].x, nHHeight - Rect[0].y, LineClr,
3);
    line->LineTo(pDC, nHWidth + Rect[1].x, nHHeight - Rect[1].y, LineClr,
3);
    line->LineTo(pDC, nHWidth + Rect[0].x, nHHeight - Rect[1].y, LineClr,
3);
    line->LineTo(pDC, nHWidth + Rect[0].x, nHHeight - Rect[0].y, LineClr,
3);
}
void CTestView::ClipBoundary(CP2i Rect0, CP2i Rect1)//窗口边界赋值函数
{
    if (Rect1.x > Rect0.x)
    {
        Wxl = Rect0.x;
        Wxr = Rect1.x;
    }
    else
    {
        Wxl = Rect1.x;
        Wxr = Rect0.x;
    }
    if (Rect1.y > Rect0.y)
    {
        Wyt = Rect1.y;
        Wyb = Rect0.y;
    }
    else
    {
        Wyt = Rect0.y;
        Wyb = Rect1.y;
    }
}
```

```
void CTestView::DrawObject(CDC* pDC, BOOL bClip)
{
    if (!bClip)
    {
        for (int i = 0; i < InMax; i++)//绘制裁剪前的多边形
        {
            if (0 == i)
                pDC->MoveTo(ROUND(nHWidth + In[i].x),
                    ROUND(nHHeight - In[i].y));
            else
                pDC->LineTo(ROUND(nHWidth + In[i].x),
                     ROUND(nHHeight - In[i].y));
        }
        pDC->LineTo(nHWidth + ROUND(In[0].x),
                    nHHeight - ROUND(In[0].y));
    }
    else
    {
        ClipPolygon(In, InMax, LEFT);
        ClipPolygon(Out, OutCount, RIGHT);
        ClipPolygon(Out, OutCount, BOTTOM);
        ClipPolygon(Out, OutCount, TOP);
        for (int j = 0; j < OutCount; j++)//绘制裁剪后的多边形
        {
            if (0 == j)
                line->MoveTo(pDC,ROUND(nHWidth + Out[j].x),
                    ROUND(nHHeight - Out[j].y));
            else
                line->LineTo(pDC,ROUND(nHWidth + Out[j].x),
                     ROUND(nHHeight - Out[j].y));
        }
        if (0 != OutCount)
            line->LineTo(pDC, nHWidth + ROUND(Out[0].x),
                     nHHeight - ROUND(Out[0].y));
    }
}
void CTestView::ClipPolygon(CP2d * out, int Length, UINT Boundary)//裁剪多边
形函数
{
    if (0 == Length)
        return;
    CP2d* pTemp = new CP2d[Length];
    for (int i = 0; i < Length; i++)
        pTemp[i]=out[i];
```

```
    CP2d p0,p1,p;//p0-起点，p1-终点，p-交点
    OutCount = 0;
    p0 = pTemp[Length - 1];
    for (int i = 0;i < Length; i++)
    {
        p1 = pTemp[i];
        if (Inside(p0, Boundary))//起点在窗口内
        {
            if (Inside(p1, Boundary))//终点在窗口内,属于内→内
            {
                Out[OutCount] = p1;//终点在窗口内
                OutCount++;
            }
            else//属于内→外
            {
                p = Intersect(p0,p1,Boundary);//求交点
                Out[OutCount] = p;
                OutCount++;
            }
        }
        else if(Inside(p1, Boundary))//终点在窗口内,属于外→内
        {
            p = Intersect(p0,p1, Boundary);//求交点
            Out[OutCount] = p;
            OutCount++;
            Out[OutCount] = p1;
            OutCount++;
        }
        p0 = p1;
    }
    delete[] pTemp;
}
BOOL CTestView::Inside(CP2d p, UINT Boundary)//在窗口边界处判断函数
{
    switch (Boundary)
    {
        case LEFT:
        {
            if (p.x >= Wxl)
                return TRUE;
            break;
        }
        case RIGHT:
        {
```

```
            if (p.x <= Wxr)
                return TRUE;
            break;
        }
        case TOP:
        {
            if (p.y <= Wyt)
                return TRUE;
            break;
        }
        case BOTTOM:
        {
            if (p.y >= Wyb)
                return TRUE;
            break;
        }
    }
    return FALSE;
}
CP2d CTestView::Intersect(CP2d p0, CP2d p1, UINT Boundary)//计算交点函数
{
    CP2d pTemp;
    double k = (p1.y - p0.y) / (p1.x - p0.x);//直线段斜率
    switch (Boundary)
    {
    case LEFT:
        pTemp.x = Wxl;
        pTemp.y = k * (pTemp.x - p0.x) + p0.y;
        break;
    case RIGHT:
        pTemp.x = Wxr;
        pTemp.y = k * (pTemp.x - p0.x) + p0.y;
        break;
    case TOP:
        pTemp.y = Wyt;
        pTemp.x = (pTemp.y - p0.y) / k + p0.x;
        break;
    case BOTTOM:
        pTemp.y = Wyb;
        pTemp.x = (pTemp.y - p0.y) / k + p0.x;
        break;
    }
    return pTemp;
}
```

```
CP2i CTestView::Convert(CPoint point)//坐标系变换函数
{
    CP2i ptemp;
    ptemp.x = point.x - nHWidth;
    ptemp.y = nHHeight - point.y;
    return ptemp;
}
```

五、实现效果

多边形 Sutherland-Hodgman 裁剪算法效果如图 20-1、20-2 和 20-3 所示。

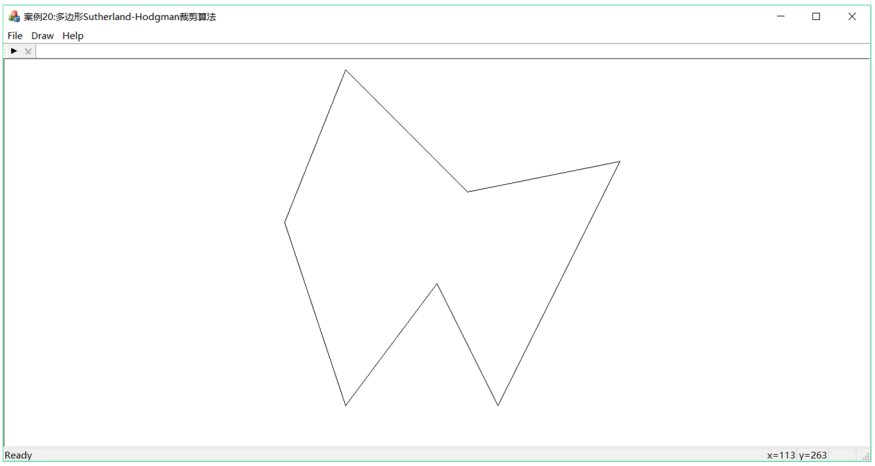

图 20-1 多边形 Sutherland-Hodgman 裁剪算法效果图一

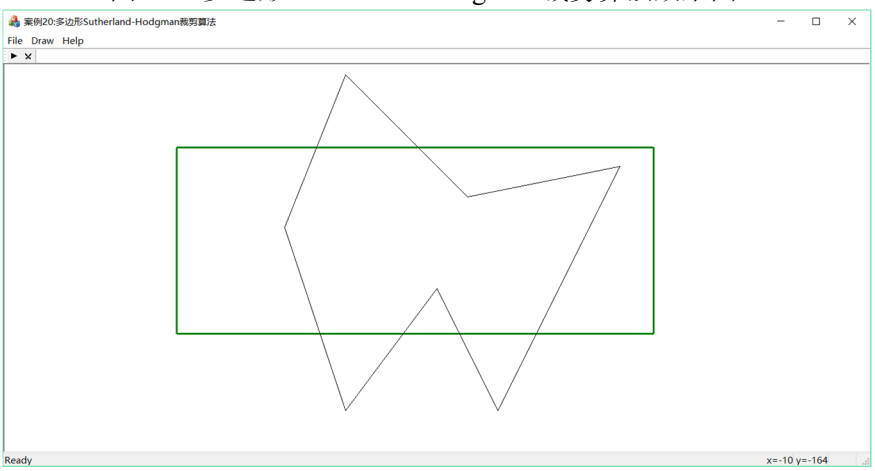

图 20-2 多边形 Sutherland-Hodgman 裁剪算法效果图二

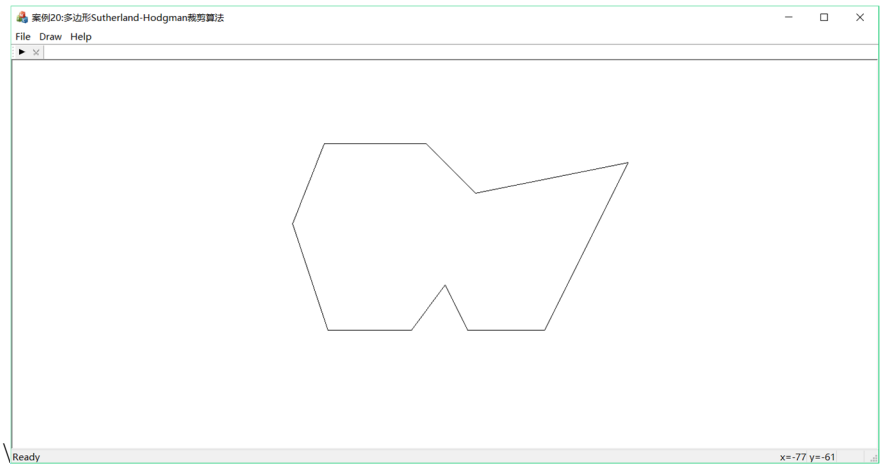

图 20-3 多边形 Sutherland-Hodgman 裁剪算法效果图三

六、知识点补充

Sutherland-Hodgman 算法在裁剪凹多边形时会出现错误，如图 20-4：

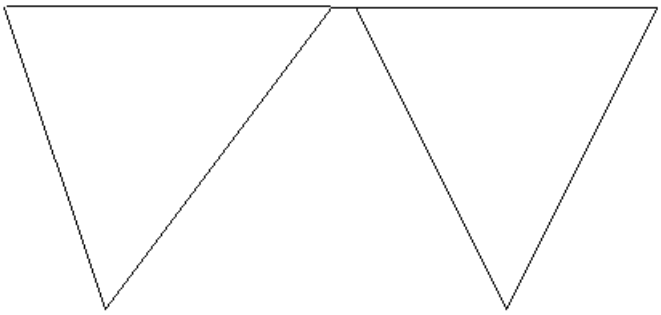

图 20-4 错误裁剪效果图

与 Sutherland-Hodgman 算法不同，weiler-Atherton 算法可以裁剪任意多边形， weiler-Atherton 算法的思路如下：

假设被裁剪多边形和裁剪窗口的顶点序列都按顺时针方向排列。当两个多边形相交时，交点必然成对出现，其中一个是从被裁剪多边形进入裁剪窗口的交点，称为“入点”，另一个是从被裁剪多边形离开裁剪窗口的交点，称为“出点”。算法从被裁剪多边形的一个入点开始，碰到入点，沿着被裁剪多边形按顺时针方向搜集顶点序列；而当遇到出点时，则沿着裁剪窗口按顺时针方向搜集顶点序列。按上述规则，如此交替地沿着两个多边形的边线行进，直到回到起始点。这时，收集到的全部顶点序列就是裁剪所得的一个多边形。由于

可能存在分裂的多边形，因此算法要考虑：将搜集过的入点的入点记号删去，以免重复跟踪。将所有的入点搜集完毕后算法结束。但是该种方法的计算量很大，效率低下，几乎不被使用。